Ausbildung Fachinformatiker Anwendungsentwicklung

Programmierübung

C–Programmierung / Datenstrukturen

1 Aufgabe

Es ist ein Programm (`shakespeare.c`) zu schreiben, das den Namen einer Text–Datei/Pfad

1. als Parameter einliest,
2. diese Datei öffnet,
3. deren Inhalt einliest,
4. bestimmte Operationen ausführt und
5. das Ergebnis in die Standard–Ausgabe schreibt.

Der Datei/Pfadname wird als Argument für die Option *-f* übergeben. Weitere optionale Parameter müssen von dem Programm eingelesen werden können. Die Parameter bestehen jeweils aus einem Buchstaben und sind durch ein vorangestelltes - gekennzeichnet. Diese Parameter bestimmen, wie die u.g. Operationen ausgeführt werden sollen. Beim optionalen Parameter `-h` soll lediglich eine `usage()`–Meldung ausgegeben werden (`./shakespeare [-h] [-s <n>] [-l <m>] [-f <inputfile>]`). Folgende Parameter müssen behandelt werden können:

Parameter	Funktion	Ausgabe
-h	mache garnichts	`usage()` ausgeben
-f <inputfile>	Datei einlesen	keine
-s <n>	Suchstringlänge	default = 1
-l <n>	Ausgabelänge	default = Anzahl eingelesener Zeichen

Das Programm soll so konzipiert werden, dass später gegebenenfalls weitere optionale Parameter leicht hinzuzufügen sind (z.B. der initiale Suchstring mit `-a <Anfang>` oder die Version mit `-v`). Außerdem soll das Programm als *Filter* arbeiten können, d.h. falls kein Datei oder Pfadname als Parameter angegeben wurde, soll das Programm seine Eingabe aus dem Kanal Standard–Eingabe (`stdin`) beziehen. Das Programm soll seine Ausgabe immer in die Standard–Ausgabe (`stdout`) schreiben. Einige mögliche Aufrufe könnten z.B. folgendermaßen aussehen:

```
bash$> ./shakespeare ./7zara10.txt
bash$> ./shakespeare -s 3 -l 1000 ./7zara10.txt
bash$> ./shakespeare -s5 -l500 ./7zara10.txt
bash$> ./shakespeare -s 8 -f ./7zara10.txt
bash$> ./shakespeare  -h
bash$> cat ./7zara10.txt | ./shakespeare -s 8
bash$> ./shakespeare -s4 -l1000 < ./7zara10.txt
```

Zum Programm soll eine etwa 4–seitige Dokumentation erstellt werden, in der die Arbeitsweise und der Aufbau des Programms kurz skizziert wird. Dabei sollte auch auf die verwendeten Datenstrukturen eingegangen werden.

Eine mögliche Ausgabe (der anfängliche Suchstring steht in spitzen Klammern) könnte ungefähr so aussehen:

```
bash$> ./shakespeare -s6 -l400 ./7zara10.txt
   <Friedr>ich Nichts; du so langsam, was auf deinem Auge. Dass du mir und Klapper mit
    Vorsicht. Aber ein Dichternen wie sieben dem als an Hoefen? Ein Arzt? Oder boese
    Blick _aller_ Dinge sie, die sich "die␣Lust.␣Lust␣nannte␣keine␣Gipfel␣zu␣neuen␣
    Frevelhaften␣Vortheil." Ungerechtig auf und hinweg und rauchten sie mich selber
    abwendet. Diess nach Vater, antwortete der Erde an und Grauen wissen auch deinetwill
bash$> ./shakespeare -s6 -l400 ./7zara10.txt
   <Friedr>ich Nichts lebt nicht es mir fragt der wieder und Entsagung. Sonderlich.
    Kein Hunger. Oft kommt er mir, so will es seiner Kuh: welche wieder _hoffende. Und
    wer werde vor Maechtigen: ‘Ja, ich mit _meine_ Sonnen-Pfeile, ein faulichkeit die
    den letzten Stuhl in die Wahrsager und tanzen macht mir dieser Vogels. Aber sein -
    und ich hart an seinem Hausthiere mieden, eine Mittel zu Ende zum
bash$> ./shakespeare -s11 -l400 ./7zara10.txt
   <Friedrich N>ietzsche
   Also sprach Zarathustra von der Stelle hin, woher die Stimme kam, und sahen zu ihm
    hinauf. Solchergestalt waren sie Alle: ihr Wahnsinn in der Liebe.
   Licht bin ich: ach, dass die Macht gnaedig wird und herabkommt in’s␣Sichtbare:␣
    Schoenheit␣waechst,␣was␣Einer␣mit␣seinem␣Blute,␣Zarathustra;␣noch␣nahmst␣du␣meinen␣
    Dank␣nicht␣an!␣Ward␣meine␣Welt␣nicht␣eben␣vollkommener␣als␣einem␣Vater,␣am␣aehn
```

2 Funktionen der Parameter

Das Programm soll einen neuen Zufallstext ausgeben, der aus dem eingelesenen Text folgendermaßen berechnet wird:

-s <n> Mit diesem Parameter wird die Suchstringlänge angegeben. Es wird nach einem Suchstring dieser angegebenen Länge im eingelesenen Text gesucht. Zuvor wird ein String dieser Länge aus dem eingelesenen Text ausgewählt (die ersten Zeichen). Dieser String ist dann der *Suchstring*.

Dann wird dieser String im gesamten Text gesucht. Der Folgebuchstabe hinter diesem String wird sich gemerkt und für jeden gefundenden Folgebuchstabe wird gezählt, wie oft er hinter diesem Suchstring im Text vorkommt. Dann wird hinter den Suchstring ein neuer Folgebuchstabe angehängt, der aus der Menge aller gefundenen Folgebuchstabe zufällig, aber unter Berücksichtigung der Wahrscheinlichkeit des Vorkommens, ausgewählt wird.

Anschließend wird bei dem um ein Zeichen verlängerte Suchstring das erste Zeichen nicht beachtet. Der neue Suchstring mit dem angefügten Zeichen hinten und dem vernachlässigten Zeichen vorne hat dann die gleiche Länge wie der anfängliche Suchstring. Mit diesem neuen Suchstring wird das Verfahren wiederholt.

Damit lässt sich ein neuer Ausgabestring erzeugen, der einen Zufallstext erzeugt, der sich aus dem eingelesenen Text zufällig ergibt.

Je kürzer der Suchstring ist, desto unsinniger wird der ausgegebene Text sein. Bei einer Länge von mehr als 5 Zeichen werden schon oft ganze Wörter berücksichtigt und der vermeintliche Sinn des

ausgegebenen Textes erscheint mehr und mehr vorhanden. Ein Beispiel für eine Suchstringlänge von 5 ($\langle abcde \rangle$) würde folgendermaßen aussehen:

$$\langle abcde \rangle_1 + [x_1] \Rightarrow \langle bcdex_1 \rangle_2 + [x_2] \Rightarrow \langle cdex_1x_2 \rangle_3 + [x_3] \ldots$$

Dabei ist $[x_i]$ jeweils ein aus der Wahrscheinlichkeitsverteilung gewählte Buchstabe, der als Folgebuchstabe zum aktuellen Suchstring $\langle \ldots \rangle_i$ im gesamten Text gefunden wurde.

-l <n> Mit diesem Parameter wird die Anzahl der Zeichen angegeben, die vom Programm ausgegeben werden sollen. Der *default*–Wert ist der gesamte Eingabetext. Falls dieser aber sehr groß ist und man nur einen Text begrenzter Länge benötigt, kann man mit diesem Parameter die Länge angeben. Der Parameter muß aber nicht unbedingt die Ausgabe begrenzen, er kann sie auch erweitern, da der generierte Text ja zufällig gebildet wird (eine Begrenzug des auszugebenden Textes sollte durch die vom Compiler größte maximal darstellbare Integer–Zahl erfolgen).

3 Bemerkungen und Tips

Die Entwicklungsumgebung ist *MinGW*. Getestet wird das Programm am günstigsten in einer *Git–bash*.

Das Einlesen der Parameter, Datei und die Verwendung als Filter sollte keine Probleme darstellen.

Eine Herausforderung stellt die Erarbeitung des Algorithmus mit den dazugehörigen Datenstrukturen dar. Daher ist es vorteilhaft, erst das Gerüst zur Behandlung der Parameter und das Einlesen der Textdatei auszuprogrammieren und zu testen.

Anschließend versuchen Sie einen Algorithmus und eine Datenstruktur zu finden, die die Anforderungen erfüllen. Bei allen Überlegungen sollte auch immer der benötigte Speicherplatz und die Laufzeit des Programmes im Auge behalten werden. Nutzen Sie Ihr Wissen über Datenstrukturen, um einen effektiven Algorithmus zu finden. Hier muss nicht nur programmiert werden, sondern es muss auch ein Konzept, die Datenstruktur und ein Algorithmus geplant werden.

Testen Sie mit einer überschaubaren kleinen Eingabedatei, bei der sich die Ausgabe leicht überprüfen lässt.

Fassen Sie die nur für die Datenstrukturen benötigten Hilfsprogramme und Datenstrukturen in externen Dateien zusammen, um die Übersicht zu behalten (Modularisierung). Da diese Aufgabe ein wenig komplexer als die vorherige ist, empfiehlt es sich hier besonders, einzelne Funktions–Komponenten gut gegeneinander abzugrenzen und separat zu testen. Nutzen Sie selbst erstellte Skizzen und Programmablaufpläne. Ein Programmieren *aus dem Bauch heraus* könnte leicht zu einem Durcheinander führen.

4 Hintergrundinformationen

Das Problem gehört zu einer Klasse von Problemen, bei dem ein Zustand durch einen festen Algorithmus immer wieder in einen neuen Zustand überführt wird. Die Menge der möglichen Zustände ist dabei fest.

Konkret ist hier die Menge der möglichen Zustände die Menge aller Möglichkeiten, einen String in der Länge des Suchstrings zu erzeugen. Bei einer Alphabetgröße z.B. von 50 ist die

$$(\text{Menge aller möglichen Zustände für einen String der Länge } s) = 50^s$$

.

Mathematisch läßt sich so ein Problem durch einen sogenannten *Markov–Prozeß* (*Markov–Kette*) beschreiben. Dabei wird aus einem Zustandsvektor (geordnete Menge aller möglichen Zustände: V_i^{alt}) durch Multiplikation mit einer Übergangsmatrix (enthält die Wahrscheinlichkeiten für die einzelnen Übergänge: P_{ij}) ein neuer Zustandsvektor (V_i^{neu}) erzeugt.) (

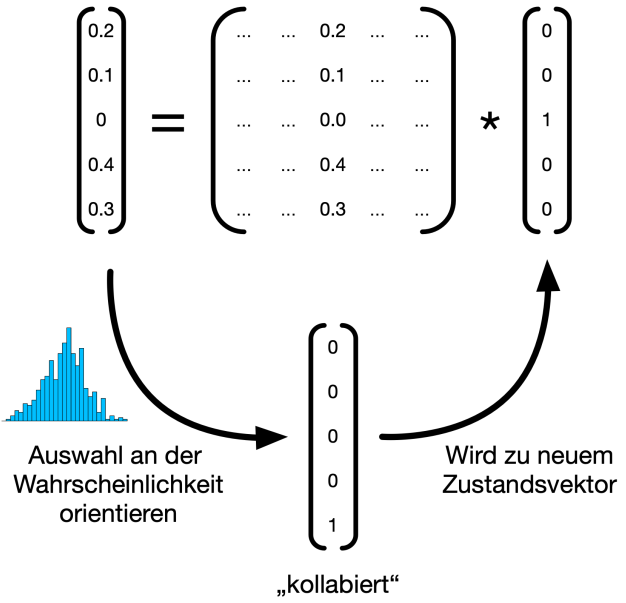
$$V_j^{\text{neu}} = P_{ij} * V_i^{\text{alt}}$$

Für dieses konkrete Problem sähen die Zustandsvektoren für einen 3–stelligen Suchstring etwa so aus:

$$V_i = \begin{pmatrix} P(AAA) \\ P(AAB) \\ P(ABA) \\ \cdots \\ P(DER) \\ \cdots \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \cdots \\ 1 \\ \cdots \end{pmatrix}$$

Aus diesem Zustandsvektor geht dann mit gewissen Wahrscheinlichkeiten ein neuer Zustandsvektor hervor. Die Wahrscheinlichkeiten sind durch die Häufigkeiten der Folgebuchstaben für den jeweiligen String definiert.

Der Index des Zustandsvektors kennzeichnet also den konkreten String. Steht dort bei V^{alt} eine 1, so ist dieser String gerade der aktuelle Suchstring. Aus diesem Zustandsvektor wird ein neuer Zustandsvektor, der bei maximal 50 Indizes (weil nur 50 verschiedene neue Buchstaben angehängt werden können) jeweils eine Zahl zwischen 0 und 1 besitzt. Diese Zahl repräsentiert die Wahrscheinlichkeit für die Wahl dieses Strings als den neuen Suchstring (alle Zahlen müssen sich daher zu 1 addieren). Mit diesen Wahrscheinlichkeiten wird ein neuer Suchstring ausgewählt (dort erscheint jetzt eine 1 und alle anderen Zeilen im Zustandsvektor werden zu 0) und der Prozess wiederholt sich. Dabei bleibt die Übergangsmatrix dieselbe, da sie nur von dem Originaltext abhängt.
Damit wäre der Algorithmus zumindest theoretisch vorgegeben.



Man müsste nur die Matrix für die Übergangswahrscheinlichkeiten aus dem Text errechnen und dann für jeden neuen Zufallsbuchstaben die Matrix mit dem Zustandsvektor multiplizieren. Dann erhält man einen neuen String mit den zugeordneten Wahrscheinlichkeiten und wählt einen gemäß der Wahrscheinlichkeit aus.

Praktisch ist das aber so nicht möglich da:

- Bei einer etwas größeren Länge für den Suchstring (wo es interessant wird: $s > 5$) wird der Zufallsvektor sehr groß. Für $s = 5$ und einem 50–Zeichen-Alphabet ergibt sich eine Zustandsanzahl von 312500000. Eine Suchstringlänge von 10 ergibt z.B. schon eine Zustandsanzahl von 97656250000000000.
- Der Zustandsvektor enthält bei einem 50–Zeichen–Alphabet nur maximal 50 Einträge ungleich Null. Damit besteht er fast nur aus Nullen.

- Die Übergangsmatrix, die die Wahrscheinlichkeiten enthält ist von der Größe des Quadrats der Größe der Zustandsvektoren und entsprechend groß.
- Abgesehen von dem astronomisch hohen Speicherbedarf würde die Berechnung bei diesen Größen entsprechend lange dauern.
- Wird der Suchstring um ein Zeichen erhöht, so würde der Speicherbedarf für den Zustandsvektor um den Faktor 50 wachsen und der Speicherbedarf für die Übergangsmatrix um den Faktor $50*50 = 2500$. Man sagt hier, dass der Algorithmus schlecht mit der Eingangsgröße *skaliert* (exponentiell). Dies wird mit der sogenannten *O–Notation* ausgedrückt, die angibt in welcher Stärke der Speicherbedarf bzw. die Laufzeit mit der Eingangsgröße wächst. Hier hätten wir ein exponentielles Ansteigen: $O(k^n), k > 2$). Für Algorithmen strebt man (abgesehen von $O(1)$) eine Komplexität von $O(log(n))$ oder $O(n)$, schlimmstenfalls $O(n^2)$ an.

Obwohl wir hier eigentlich einen klaren Algorithmus hätten , der einfach ausprogrammiert werden könnte (und auf 1– und 2–dimensionalen Feldern/Arrays beruht), müssen wir nach einem praktischeren Algorithmus suchen, der mit dem zur Verfügung stehenden Speicher auskommt und ihn effizienter nutzt!

5 Abgabe

Die lauffähigen Programme und der kommentierte Quelltext sind mit einer etwa 4–seitigen Dokumentation abzugeben (Email, Dokumentation in HTML,TEX oder `ASCII`).

6 Zeitrahmen

Der Zeitrahmen dieser Aufgabe ist mit etwa 60 Stunden angesetzt.